integrata
Cegos Group

inspire
qualify
change

Programmierung mit .NET

Kapitel 1

EINFÜHRUNG IN .NET

Was ist .NET? (1/3)

integrata Cegos Group
inspire qualify change

- Virtuelle Maschine
  - Isoliert Programm von Hardware und Betriebssystem
  - Ähnlich Java, aber sprachübergreifend
  - Keine Auswirkung auf Programmoberfläche
- Standardisiert von ECMA und ISO

Was ist .NET? (2/3)

integrata Cegos Group
inspire qualify change

- Zu .NET zählt Microsoft
  - .NET Framework und .NET Core, bilden Laufzeitumgebung
  - Entwicklungsumgebung Visual Studio
  - Server-Software wie Windows Server, SQL Server und BizTalk Server
  - Client-Software wie Windows, Windows Phone und Microsoft Office, aber auch Linux und OS X

Was ist .NET? (3/3)

integrata Cegos Group
inspire qualify change

- Zum .NET Framework zählen u.a.:
  - ASP.NET
    - ASP.NET Forms
    - ASP.NET MVC
  - ADO.NET
  - Windows Forms
  - Windows Presentation Foundation (WPF)
  - Windows Communication Foundation (WCF)
  - Windows Workflow Foundation
  - Windows Identity Foundation
  - LINQ
  - Parallel Extensions
  - Entity Framework

- Zu .NET Core zählen:
  - ASP.NET Core
  - Entity Framework Core

Warum .NET? (1/5)

integrata Cegos Group
inspire qualify change

- Das Ende der DLL-Hell
  - Löschen oder Überschreiben gemeinsam genutzter DLLs
  - COM-DLLs konnten nur einmal auf Rechner installiert sein
  - .NET: Registry überflüssig
  - .NET ermöglicht parallele Installation mehrerer DLL-Versionen
  - .NET-Programm enthält Informationen über benötigte DLLs
  - .NET: Umbenennen / Verschieben v. Dateien / Verzeichnissen OK

Warum .NET? (2/5)

- Vereinfachte Installation, Konfiguration und Deinstallation
  - .NET-Programme: keine Registry-Einträge
    -> XCopy-Bereitstelllung
  - ClickOnce-Bereitstellung
  - .NET: Konfigurationsdaten in XML-Datei im Programmverzeichnis
    - Leicht zu vergleichen / versionieren
    - Leicht zu ändern, auch per Skript
  - .NET: Zusätzlich Konfigurationsdateien auf Rechner- oder Unternehmensebene möglich
    -> Admin freut sich
  - .NET: Deinstallation = Verzeichnis löschen

Warum .NET? (3/5)

integrata Cegos Group
inspire qualify change

- Neue Infrastruktur für verteilte Programme
  - COM / DCOM nur im Intranet geeignet
  - COM außerhalb von Windows kein Standard
  - Konkurrenz durch Java
  - Windows Communication Foundation (WCF)
  - .NET Web Services = plattformübergreifender Standard
  - WebAPI
  - Quelloffenenes .NET Core

Warum .NET? (4/5)

integrata Cegos Group
inspire qualify change

- Einheitliches Programmiermodell
  - Windows-Programmierung historisch gewachsen
    → Probleme, z. B.
    - Fehlerbehandlung uneinheitlich
    - Unterschiedl. Typen + Aufrufkonv. der Programmiersprachen
  - .NET: einheitliches Programmiermodell
    - Einheitliche, objektorientierte Bibliothek
    - Vereinheitlichte Fehlerbehandlung
    - Einheitliches, erweiterbares Typsystem
    - Einarbeiten in Programmiersprache = Syntax lernen

Warum .NET? (5/5)

integrata Cegos Group
inspire qualify change

- Mehr Sicherheit
  - Programmierer verwalten Speicher -> Probleme
    - Speicherlecks
    - Buffer-Overrun-Angriffe
  - Speichermanipulationen durch Zeiger -> mehr Probleme
    - Programmierer evtl. überfordert
    - Bösartiger Code möglich
  - .NET: alles wird gut
    - Garbage Collector = keine Speicherlecks
    - .NET-Sprachen typsicher, keine Zeiger = kein bösartiger Code
    - Rollenbasierte + Codezugriffssicherheit auf allen Betriebssystemen

Verfügbarkeit des .NET Framework (1/2)

integrata Cegos Group
inspire qualify change

- .NET Framework
  - In aktuellen Windows-Versionen eingebaut
  - Mit Visual Studio .NET installiert oder von Zeitschriften-CD
  - .NET Framework SDK (+ #Develop, Matrix, ...)

Verfügbarkeit des .NET Framework (2/2)

integrata Cegos Group
inspire qualify change

- MS: Shared Source CLI mit Quelltext
  - FreeBSD, Mac OS X
  - Keine kommerzielle Verwendung
- (Novell:) Mono-Projekt mit Quelltext
  - Linux, Windows, Mac OS X, FreeBSD und Sun Solaris
  - Open-Source-Projekt, Laufzeitumg. + Entwicklungswerkz. + ...

Nutzung des .NET Framework (1/4)

integrata Cegos Group
inspire qualify change

- .NET-Programme schreiben
  - Framework Class Library (FCL) = Funktionsbibliothek
  - Zugriff auf Windows-API oder COM-Komponenten möglich (aber Sicherheit + Performance leiden)
  - Gleiche Plattform auf allen Systemen

Nutzung des .NET Framework (2/4)

integrata Cegos Group
inspire qualify change

- .NET-Programme installieren
  - XCopy-Bereitstellung
  - .EXE, .DLL = Intermediate Language (IL) + Metadaten
    - Plattformunabhängigkeit
    - Disassemblierbar, Problem?
  - Assembly
    - Enthält Beschreibung benötigter Bibliotheken und Rechte
    - Besteht aus einer oder mehreren Dateien
    - Konfigurationsmöglichkeiten unabhängig vom Betriebssystem
    - Global Assembly Cache (GAC)

Nutzung des .NET Framework (3/4)

integrata Cegos Group
inspire qualify change

- .NET-Programme ausführen >
  - Grafik im Kapitel *.NET-Referenz*
  - In Common Language Runtime (CLR) laden
    - Metadaten in Assembly -> Bibliotheken, Rechte
    - Konfigurationsdateien auswerten
  - Just-in-Time kompilieren (JITten)
    - IL -> Betriebssystemspezifischer Code
    - Sichtbarkeit + Typ von Variable bekannt -> Garbage Collection möglich
    - CLR erkennt Programmabsicht -> kann abbrechen
    - Zeiger, z. B. in C# -> unsicherer Code, braucht höchste Ausführungsrechte
    - API-Funktionen, COM-Komponenten → unverwalteter Code, noch schlimmer

Nutzung des .NET Framework (4/4)

integrata Cegos Group
inspire qualify change

- .NET-Programme ausführen
  - Anwendungsdomäne
    - Isoliert wie Prozess -> Sicherheit
    - Leichter als Prozess -> Geschwindigkeit, Sparsamkeit
    - Auch auf Betriebssystemen ohne Prozesse möglich
  - CLR-Host
    - Nötige Erweiterung für Programm / Betriebssystem
    - Entscheidet Verhältnis .NET-Programm / Anwendungsdomäne / Prozess
    - MS liefert CLR-Hosts für Windows, IIS und IE
    - Mono-Projekt liefert CLR-Hosts für Windows, FreeBSD, Mac OS X, Linux + für Apache-Webserver

Kapitel 2

VISUAL STUDIO

Was ist Visual Studio ?

integrata Cegos Group
inspire qualify change

- Integrierte Entwicklungsumgebung (IDE), unterstützt
  - Das, was alle IDEs unterstützen sowie
  - Entwickeln in mehreren Programmiersprachen
  - Erstellen von Web-Oberflächen
  - Erstellen von Windows Apps
  - Entwickeln von Programmen für mobile Geräte
  - Manipulieren von HTML, CSS und XML
  - Einbinden + grafisches Manipulieren von Datenquellen incl. Debugging von Stored Procedures im SQL Server

Visual Studio installieren

integrata Cegos Group
inspire qualify change

- Visual Studio installieren
  - Zeit mitbringen, dauert ca. 1h
  - HD-Platz kontrollieren, 4-10 GB mind. 5400 RPM, teilweise nur temporär nötig
  - MS Empfehlungen: PC >= 1.6 GHz, >= 1024 MB RAM
  - MS SQL Server LocalDB
  - Mind. Windows 7 SP1 oder Windows Server 2008 R2 SP1

Gemeinsam

STARTEN VON VISUAL STUDIO

Visual Studio starten (1/3)

integrata Cegos Group
inspire qualify change

- Globale Einstellungen
- Erweiterungen
- Neues Projekt, Sprachen
  - Visual Basic
  - Visual C#
  - F#
  - Visual C++ (C++ with Managed Extensions)
  - ...

Visual Studio starten (2/3)

integrata Cegos Group
inspire qualify change

- Neues Projekt, Typen (sprachabhängig und auszugsweise, hier für C#)
  - Windows-Anwendung
  - Klassenbibliothek
  - Windows-Steuerelementbibliothek
  - Web-Steuerelementbibliothek
  - Konsolenanwendung
  - Windows-Dienst
  - ASP.NET Webanwendungen
  - ASP.NET Core Anwendungen
  - Azure
- Neue Website, Typen
  - ASP.NET Webanwendung
  - ASP.NET Webdienst
  - ASP.NET Mobile-Webanwendung
  - Azure

Visual Studio starten (3/3)

integrata Cegos Group
inspire qualify change

- Speicherort wählen
  - Standard-Speicherort
  - Automatisch zusätzl. Unterverzeichnis = Projektname
  - Integration des Standard-Speicherorts in Datensicherung bedenken
  - Später vorzuschlagenden Standard-Speicherort festlegen
  - Framework Version

Die wichtigsten Fenster von Visual Studio (1/3)

integrata Cegos Group
inspire qualify change

- Designer-Fenster, auch für WebForms-Oberflächen!
  - Zum Gestalten der grafischen Oberfläche
  - Werkzeugleisten automatisch eingeblendet
- Toolbox
  - Enthält Komponenten und Steuerelemente
  - Sichtbare Registerkarten abhängig vom Designer
- Eigenschaften-Fenster
  - Zeigt Eigenschaften des gewählten Steuerelements
  - Unterschiedliche Sortierung möglich
  - Umschalten auf Ereignisanzeige möglich

Die wichtigsten Fenster von Visual Studio (2/3)

integrata Cegos Group | inspire qualify change

- Projektmappen-Explorer
  - Zeigt Baumdarstellung des Projekts
  - Mehrere Projekte pro Projektmappe möglich
  - Umschalten zwischen Design- und Code-Ansicht möglich
  - Kann ausgeblendete Elemente des Projekts anzeigen
  - Objektbrowserfunktionalität
  - Suchen im Explorer
  - Projektmappenordner für logische Struktur
- Code-Editor, bietet
  - Syntax-Highlighting
  - Codegliederung
  - Codekommentare
  - TODO-Kommentar, etc.
  - Ausfüllhilfen

Die wichtigsten Fenster von Visual Studio (3/3)

integrata Cegos Group
inspire qualify change

- Online-Hilfe
  - F1 über Alles
  - Viele Beispiele
  - Jetzt im externen Fenster
  - Neugierig sein, weiterhangeln -> nach 1 Jahr .NET-Guru

Visual Studio konfigurieren

integrata Cegos Group
inspire qualify change

- Zentraler Arbeitsbereich
  - Fenster über Registerkarten sortierbar
  - Ganzer Bildschirm möglich
- Hilfsfenster
  - Beliebig positionierbar und andockbar
  - Fensterlayout in Installationszustand zurücksetzbar
  - Automatisches Ein- und Ausblenden der Toolbox abstellbar
- Weitere Features
  - Docking-Mechanismus
  - Projektmappenverzeichnis automatisch erstellt
  - Automatisches Speichern geänderter Dateien
  - Einstellungen ex- / importieren

Steuerelemente und Bibliotheken einbinden (1/2)

integrata Cegos Group
inspire qualify change

- Steuerelemente einbinden
  - Rechtsklick auf Registerkarte der Toolbox
  - Aufgelistete Steuerelement-Bibliotheken wählen oder durchsuchen
  - Steuerelement mit Entfernen-Taste löschen oder
    Häkchen vor Steuerelement-Bibliothek entfernen

Steuerelemente und Bibliotheken einbinden (2/2)

integrata Cegos Group
inspire qualify change

- Bibliotheken einbinden
  - Verweis hinzufügen (Hauptmenü oder Projektmappen-Explorer)
  - DLL wählen, erscheint im Projektmappen-Explorer
  - using-Direktive / Alias-Name nicht vergessen!
  - Mit Entfernen-Taste im Projektmappen-Explorer aus Projekt entfernen
  - Suchfunktionalität im Dialog

Rapid Application Development mit Visual Studio (1/5)

integrata Cegos Group
inspire qualify change

- Rapid Application Development (RAD)
  - Grafisches Erstellen der Oberfläche aus Steuerelementen
  - Konfigurieren der Steuerelemente durch Einstellen ihrer Eigenschaften
  - Schreiben von Code für Funktionalität, die den Steuerelementen fehlt

Rapid Application Development mit Visual Studio (2/5)

integrata Cegos Group
inspire qualify change

- Oberfläche grafisch erstellen
  - Grundlegende Gestaltungsregeln
    - Möglichst wenig Fensterwechsel
    - Alle Befehle über Menüs erreichbar
    - Deaktivieren, nicht unsichtbar machen
    - Rückkopplung geben
    - Fenster nach einheitlichem Muster aufbauen, z.B. wie Explorer oder Outlook
    - Anordnung der Steuerelemente entsprechend dem Arbeitsfluss von links nach rechts und von oben nach unten
    - Einheitliches Layout aller Fenster, z.B. Abstand zwischen Steuerelementen und Ausrichtung der Beschriftung zum Steuerelement
    - Tastaturbedienung (nicht nur) für Vielschreiber
    - .NET unterstützt visuelle Vererbung!

Rapid Application Development mit Visual Studio (3/5)

integrata Cegos Group
inspire qualify change

- Oberfläche grafisch erstellen
  - Steuerelemente hinzufügen
    - Umschalten in Designer-Ansicht mit F7 bzw. Umsch+F7 oder Schaltflächen oberhalb des Projektmappen-Explorers
    - Hinzufügen durch Doppelklicken oder durch Drag & Drop aus Toolbox
    - Container-Steuerelemente beachten
    - Steuerelemente zur Laufzeit dynamisch erzeugen oder verschieben meistens keine gute Idee
  - Steuerelemente markieren und ausrichten
    - Einfach- und Mehrfachmarkierung
    - Werkzeugleiste Layout
    - Steuerelemente evtl. gegen Verschieben sperren

Rapid Application Development mit Visual Studio (4/5)

integrata Cegos Group
inspire qualify change

- Steuerelemente konfigurieren
  - Eigenschaften im Eigenschaften-Fenster setzen oder zur Laufzeit
  - Steuerelement-Auswahl im Formular oder über Listenfeld, Mehrfachauswahl möglich
  - Einstellungen von Container-Steuerelementen haben evtl. Auswirkungen auf untergeordnete Steuerelemente
  - Werte automatisch zu Methode `InitializeComponent()` in *.Designer.cs-Datei generiert; besser **nicht** manuell verändern!

Rapid Application Development mit Visual Studio (5/5)

integrata Cegos Group
inspire qualify change

- Code schreiben
  - Ereignisse behandeln, im Eigenschaften-Fenster sichtbar
  - Ereignishandler haben immer 2 Parameter: `sender` + `EventArgs`
  - Verbindung zwischen Handler + Ereignis in `InitializeComponent()`-Methode; Delegat-Konzept!
  - Ereignis im Eigenschaften-Fenster mit vorhandenem Ereignishandler verbinden
  - Verbindung zum Ereignishandler im Eigenschaften-Fenster aufheben
  - Verbindung im Code setzen möglich, dort auch Ereignis mit mehreren Handlern verbinden möglich
  - Shortcut für Click Events

Projekteigenschaften verwalten

integrata Cegos Group
inspire qualify change

- Projekteigenschaften verwalten
  - Projekt-Designer
  - Erspart Kommandozeilenwerkzeuge, Editieren von Konfigurationsdateien
  - Registerkarten

Sonstige Features in Visual Studio

integrata Cegos Group
inspire qualify change

- Sonstige Features in Visual Studio
  - Refactoring-Unterstützung
  - Klassenansicht, Klassendiagramm (Roundtrip-Engineering!)
  - Build-Werkzeug
  - Team Edition
    - Codeanalyse
    - Code-Coverage-Analyse
  - Weiterhin gut ...
    - NUnit
    - NProf
  - Multitargeting
    - Anlage/Bearbeitung von Projekten mit unterschiedlichen .NET Framework Versionen
  - Suchen nach Codeklonen

Kapitel 3

KONSOLENANWENDUNGEN

Zweck einer Konsolenanwendung

integrata Cegos Group
inspire qualify change

- Vereinfacht das Programmieren
- Konsole fehlt aber z. B. auf Windows Phone
- Heute meist nur noch für Test- oder Demonstrationsprogramme
  - RAD erleichtert Programmieren grafischer Oberflächen
  - Nutzer sind Besseres gewöhnt
  - Programme für Programmierer

Anatomie einer Konsolenanwendung

integrata Cegos Group | inspire qualify change

- Vorlage Konsolenanwendung in Visual Studio
- *`Program.cs`*, Name änderbar
- Elemente im Projektmappen-Explorer ansehen
- Dateien und Verzeichnisse im Explorer ansehen
- Beschreibung der Dateien und Verzeichnisse siehe Tabelle
- *`Program.cs`*:
  - Namensraum = Projektname
  - Klasse `Program`
  - Methode `Main()`
  - Eigene Felder + Definitionen auf Klassenebene oberhalb `Main()`

Funktionsweise einer Konsolenanwendung

integrata Cegos Group | inspire qualify change

- Start -> `Main()`
- Ende `Main()` -> Ende Programm
- Kommandozeilenargumente in `args` Parameter
  - 1. Parameter nicht Anwendungsname
- `Main():int` statt `void` möglich
- Kommunikation über `Console`-Klasse
- Standard-Eingabe, -Ausgabe und -Fehlerausgabe umleitbar
  - `StartInfo`-Eigenschaft der `Process`-Klasse
  - `StandardInput`-, `StandardOutput`- und `StandardError`-Eigenschaften

Eingaben auswerten und Text ausgeben

integrata Cegos Group
inspire qualify change

- `Console`-Klasse aus Namensraum System
- Methoden `ReadLine()`, `Read()`, `WriteLine()` und `Write()`
- Lesen in Schleife
- Mit `ReadLine()` auf Eingabe des Benutzers warten

Kommandozeilenparameter auswerten

integrata Cegos Group
inspire qualify change

- Kommandozeilenparameter in `Main()` im Parameter `args`
- `args`: String-Array mit d. Leerzeichen getrennten Teilen der Kommandozeile
- Kommandozeilenparameter beim Aufruf aus IDE:
  - Projekt, Eigenschaften, Debuggen, Startoptionen, Befehlszeilenargumente

Features für Konsolenanwendungen in .NET

- Vorder- + Hintergrundfarbe des Textes einstellbar
- Ausgabepuffer (= bestehende Ausgabe) verschieben / löschen
- Position, Größe, Titelleistentext des Konsolenfensters per Code änderbar
- Noch mehr kosmetisches ...

Kapitel 4

PROGRAMMIEREN MIT DEM .NET FRAMEWORK

Warum C#?

integrata Cegos Group | inspire qualify change

- Microsoft beginnt Mitte der 90er Jahre, .NET zu entwickeln
  - Bestehende Sprachen für spezielle Compiler, Bibliotheken und BS entwickelt
    → passten nicht zu .NET
  - Bestehende Sprache umzuarbeiten rechtlich nur für VB möglich
    -> Probleme mit alten VB-Programmierern
- Also technische / juristische Notwendigkeiten
  → C# nichts Neues, Best-of bekannter OO-Programmiersprachen
- C#
  - Features von Delphi durch Anders Hejlsberg
  - Folgt seinem Namen entsprechend C-ähnlicher Syntax
  - Wenig gemeinsam mit Sprache Java; .NET viel gemeinsam mit Plattform Java
  - „ßi-scharp“ gesprochen
  - C# ist um einen Halbton erhöhtes C -> C# Weiterentwicklung von C
  - C# → Syntax möglicher Sprachkonstrukte
  - (.NET → Typsystem + Klassenbibliothek aller Sprachen)

Ein einfaches C#-Programm (1/2)

integrata Cegos Group
inspire qualify change

- Hallo Welt!

```
class Min {
     static void Main() {
          System.Console.WriteLine("Hallo Welt!");
     }
}
```

Ein einfaches C#-Programm (2/2)

integrata Cegos Group | inspire qualify change

- Programm zeigt:
  - Keine globale Funktionen oder Variablen
  - (Nur wenigen Definitionen außerhalb einer Klasse möglich)
  - C# keine Hybridsprache wie C++ oder Delphi
  - Methode `Main()`
    - Außer in DLLs immer nötig
    - Andere Signatur möglich
    - Zugriffsmodifizierer wie `private` und `public` werden ignoriert
    - C# unterscheidet Groß- und Kleinschreibung, `Main()` != `main()`
    - .NET-Klassenbibliothek in Namensräume untergliedert, immer angeben

Besonderheiten von C# (1/2)

integrata Cegos Group | inspire qualify change

- Wenig Überraschungen, nur Details
  - Keine globale Funktionen oder Variablen
  - Keine lokalen statischen Variablen, private Felder benutzen
  - Lokale Variablen werden nicht automatisch initialisiert, nur Felder
  - Groß- und Kleinschreibung wird unterschieden
  - Umlaute in Bezeichnern erlaubt, aber keine Sonderzeichen wie $ oder #
  - Lokale Variablen lassen sich blockweise deklarieren, z. B. in `if`-Block
  - C# definiert Alias-Namen für FCL-Typen, z. B. `int` für `System.Int32`
  - Alles ist ein Objekt, auch einfachere Typen wie Integer
  - C# / .NET Framework unterscheidet zwischen Wert- und Referenztypen
  - Strukturen können Methoden + Ereignisse enthalten + Schnittstellen implementieren
  - Partielle Klassen, Schnittstellen, ...
  - Generische Klassen, Schnittstellen, ...
  - `Nullable` Types (Werttypen, die den Wert `null` annehmen können)
  - Autoimplemented Properties
  - Delegaten

Besonderheiten von C# (2/2)

integrata Cegos Group
inspire qualify change

- Sichtbarkeit lokaler Variablen durch Ort der Deklaration bestimmt
- Sichtbarkeit von Feldern durch Zugriffsmodifizierer definiert
  - `public`: Uneingeschränkter Zugriff
  - `protected`: Zugriff innerhalb des Typs und davon abgeleiteter Typen
  - `internal`: Zugriff innerhalb der Assembly
  - `protected internal`: Innerhalb Assembly wie `public`, außerhalb wie `protected`
  - `private`: Zugriff innerhalb des Typs

Wert- und Referenztypen (1/3)

integrata Cegos Group
inspire qualify change

- Werttypen
  - Kleine, kurz benutzte Typen wie Integer, Aufzählungen und Strukturen
  - Schnell zugreifbar
  - Variablen enthalten Wert selber
- Referenztypen
  - Größere, langfristig verwendete Typen wie Klassen und Strings
  - Variablen enthalten nur Verweis auf eigentlichen Wert
  - Garbage Collector, keine Freigabe nötig

Wert- und Referenztypen (2/3)

integrata Cegos Group
inspire qualify change

- Vergleichen von Referenztyp-Variablen
  - Verglichen werden Referenzen, nicht Objekte selber
  - `Equals()` + `ReferenceEquals()`, Vorsicht bei Basisimplementierung
- Zuweisen von Referenztypen an Variablen
  - Kopie der Referenz wird übergeben
  - Kopie der Referenz verweist auf dasselbe Objekt wie Original
  - Veränderungen wirken sich direkt auf das Objekt aus

Wert- und Referenztypen (3/3)

integrata Cegos Group
inspire qualify change

- Zuweisung erzeugt evtl. im Hintergrund aus Werttyp einen Referenztyp
  - Boxing
  - Beim Zugriff dann umgekehrter Weg - Unboxing
  - Beides automatisch, kostet aber Performance
  - Generische Auflistungsklassen
- Strings
  - Sind Referenztypen
  - Verhalten sich aber wie Werttypen
    - Verglichen werden ihre Werte
    - Beim Zuweisen werden sie kopiert
- Strukturen
  - Große Strukturen -> aufwändige Kopieroperationen beim Zuweisen
  - Statt großer Strukturen entsprechende Klassen einsetzen

Strings

- Zeichen in .NET
  - Unicode
  - Zeichenketten: System.String, z. B. "Ein String"
  - Einzelne Zeichen: System.Char, z. B. 'c'
- System.String
  - Referenztyp mit Werttyp-Semantik
    - String-Vergleich -> Vergleich der Werte, kein Vergleich der Referenzen
    - Verkettung möglich -> Kopieroperationen, **StringBuilder** benutzen!
  - Compare(), Equals(), StartsWith(), IndexOf()
  - Zugriff auf Zeichen mit Array-Syntax
  - Format() ersetzt Platzhalter in String durch formatierte Werte
  - Zeilenumbruch in String durch \r\n oder Environment.NewLine
  - Rückstrich zugleich Escape-Zeichen
    - string s = "c:\\";
    - string s = @"c:\";

Typumwandlung und -abfrage

integrata Cegos Group | inspire qualify change

- C# / .NET Framework ist typsicher -> Compiler findet viele Fehlerquellen
- Implizite Typumwandlung, z. B. von `Int32` auf `Int64`, automatisch
- Explizite Typumwandlung
  - Harte Typumwandlung, z. B. `(Int32)i` – Laufzeitfehler möglich!
  - Typsichere Umwandlung mit ***`as`*** – ***`null`***, wenn Umwandlung fehlschlägt
- Typabfrage
  - Operator **`is`**, z. B. **`if (i is Int32)`** ...;
  - Type-Objekt
    - Instanz: Methode `GetType()`
    - Typ: `typeof()`-Operator
- Typen wie z. B. `System.Boolean` und `System.Int32` besitzen statische Methode `Parse()` zum Instanziieren aus `String`

Aufzählungen (Enum)

- Aufzählung
  - Satz von Konstanten
  - Typsicherheit
  - Selbstdokumentierender Code
  - `Enum`–Klasse bietet verschiedene Methoden zum Auswerten
  - Als Bitfeld möglich

Schnittstellen

integrata Cegos Group | inspire qualify change

- .NET Framework und seine Schnittstellen
- Lassen sich von anderen Schnittstellen ableiten
  - Erben Funktionalität d. Basisschnittstelle
  - Fördern Planung, Organisation + Wiederverwendbarkeit d. Codes
- Schnittstellen und Member müssen `public` oder `internal` sein
  - Schnittstellen dienen zur Veröffentlichung der öffentlich sichtbaren Member
  - Schnittstellen unterstützen alle Member-Typen **außer** Feldern
- Schnittstellen sind die Blaupause der Klassenstruktur
- Klassen realisieren die Schnittstellen
  - Klassen können beliebig viele Schnittstellen realisieren
- `interface`-Schlüsselwort
- Eine bewährte und empfohlene Vorgehensweise bei der .NET-Entwicklung ist: Schnittstelle → Testklasse → Klasse
- Gemäß Best Practice bekommen Schnittstellennamen immer den Buchstaben `I` vorangestellt

Strukturen

- Fassen Typen zu thematisch zusammenhängender Einheit zusammen
- Können Methoden + Schnittstellen implementieren + Ereignisse auslösen
- Einschränkungen gegenüber Klassen
  - Basieren auf Typ `struct`, können nicht Basis einer Ableitung bilden
- Sind Werttypen
  - Müssen dadurch bei Parameter-Übergabe kopiert werden
  - Nur für kleine Typen nutzen
  - Größere Typen als Klassen, wenn keine Werttyp-Semantik nötig
- Weitere wichtige Eigenschaften
  - Strukturen sind Nullable Types
  - Generische Strukturen

Klassen

integrata Cegos Group
inspire qualify change

- Funktionalität wie Strukturen
- Lassen sich von anderen Klassen ableiten
  - Erben Funktionalität d. Basisklasse
  - Fördern Organisation + Wiederverwendbarkeit d. Codes
  - Eine Klasse kann immer nur **eine** Klasse zur gleichen Zeit erben
- Sind Referenztypen
  - Parameterübergabe speichereffizienter
- `class`-Schlüsselwort
- Optional Basisklassenangabe und `implements`
- Nicht automatisch instanziiert, **`new`** benutzen!

Generische Typen

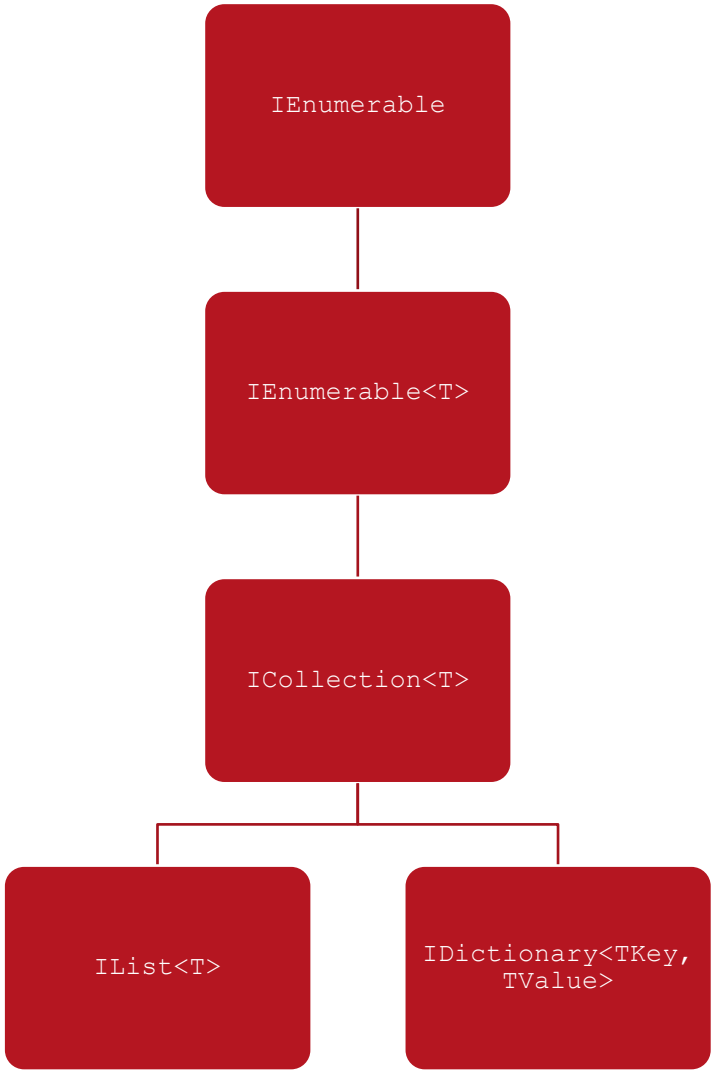
integrata Cegos Group
inspire qualify change

- Vorteile generischer Typen
  - Typsicher
  - Keine Typumwandlungen
  - Kein Boxing
- Mögliche generische Typen können eingeschränkt werden durch
  - where T : <IMyInterface>
  - where T : <MyBaseClass>
  - where T : U
  - where T : new()
  - where T : struct
  - where T : class

Arrays

- Arrays
  - Werden zur Aufnahme eines bestimmten Typs deklariert
    - Kein Boxing und Unboxing für Werttypen
    - Aber: generische Auflistungsklassen
  - Mehrdimensionale und unregelmäßige Arrays möglich
  - Keine dynamischen Arrays
  - Unterstützen `foreach`-Schleife
  - Untere Grenze festlegbar, aber lieber bei 0 bleiben
  - Sind Referenztypen
    - Zuweisung kopiert nur Verweis auf Array, nicht dessen Elemente
    - `Copy()`, `CopyTo()` und `Clone()`
    - Trotzdem enthält Array-Kopie Verweise auf **dieselben** Instanzen von Referenztypen
  - Zum Sortieren: `Sort()`, `Reverse()`
  - Zum Suchen: `BinarySearch()`, `IndexOf()`
  - Anzahl der Elemente: `Length`, `GetLength()`
  - `IndexOutOfRangeException`

Auflistungen (1/2)



Auflistungen (2/2)

integrata Cegos Group
inspire qualify change

- Namensräume `System.Collections` und `System.Collections.Specialized`
- Auflistungsklassen wie `Queue`, `Stack`, `ArrayList`, `HashTable` und `NameValueCollection`
- Spezialisierter Funktionsumfang
- Können dynamisch wachsen
- Elemente als `object` verwaltet
  - Referenztypen OK
  - Werttypen führen zu Boxing und Unboxing, nachteilig für Performance
- Auswählen einer Auflistungsklasse → Online-Hilfe
- Namensraum `System.Collections.Generic`
  - Typsicher
  - Zur Laufzeit entsteht automatisch optimierter Code

Ereignisse

- Ereignisse
  - Zum Benachrichtigen eines oder mehrerer Empfänger
  - Z. B. Click-Ereignis der Button –Klasse
  - Empfänger implementiert Handler
  - Verbindung durch Delegat
    - Objekt
    - Signatur + Referenz auf Empfänger + dessen Rückrufmethode
    - Typsicher, Compiler findet Fehler
    - Liste von Rückrufmethoden
- Empfohlene Ereignis-Signatur unter .NET
  - Sender vom Typ `object`
  - Zusatzinformationen vom Typ `EventArgs` oder abgeleiteter Klasse

Ausnahmen

integrata Cegos Group
inspire qualify change

- Laufzeitfehler
  - NET Framework erzeugt Ausnahme
  - Programm in undefiniertem Zustand
- Ausnahmebehandlung >
  - Ohne Behandlung im Programm bricht es die CLR i. A. ab
  - Trotzdem Ausnahme nur behandeln, wenn Programm wieder in definierten Zustand gebracht werden kann
  - Auch behandelte Ausnahmen protokollieren
    - Exception Management Application Block
    - FCL fängt leider viele Ausnahmen ab
- Exception-Klasse
  - Ausnahme ist Instanz von Exception oder abgeleiteter Klasse
  - Exception-Klasse bietet zusätzliche Informationen, z. B. Aufrufliste
  - Von FCL-Ausnahmen eigene Klassen ableitbar, siehe Online-Hilfe

Ausnahmebehandlung

integrata Cegos Group
inspire qualify change

- `try` ... `catch`-Anweisung
  - Gefährdete Anweisungen in `try`-Block fassen
  - Evtl. aufgetretene Ausnahme im folgenden catch-Block verarbeiten
- `catch`-Block
  - Gibt Ausnahme automatisch frei
  - Wird nur ausgeführt, wenn Ausnahme aufgetreten ist
  - Programmausführung geht in Anweisung nach `catch`-Block weiter
  - Typ der zu behandelnden Ausnahme angeben, schließt abgeleitete Typen ein
  - Nicht Exception = generell alle Ausnahmen behandeln
  - Weitere `catch`-Blöcke möglich, spezifische zuerst
- `throw`-Anweisung
  - Ausnahme im `catch`-Block wieder auslösen
    - ursprünglichen Aufrufliste geht verloren -> `InnerException` setzen
  - Ausnahmen generell auslösen, dazu Instanz erzeugen

Ressourcenschutz-Blöcke

integrata Cegos Group
inspire qualify change

- `try` … `finally`- / `using`- Anweisung
  - Gefährdete Anweisungen in `try`-Block fassen
  - Im `finally`-Block unverwaltete Ressource freigeben
  - `finally`-Block wird immer durchlaufen, auch bei `return`
  - `using`-Anweisung noch einfacher
  - `finally`-Block / `using`-Anweisung gibt Ausnahme nicht frei

Ziele des objektorientierten Programmierens

integrata Cegos Group
inspire qualify change

- Objektorientierung
  - Ermöglicht, reale Objekte und Vorgänge im Programm abzubilden
  - Programmierer kann auf niedrigerer Abstraktionsstufe arbeiten
  - Resultat kontinuierlicher Entwicklung
    - Einfachster Datentyp: Binärsystem
    - Praktischere Datentypen: String, Integer
    - Noch praktischere Datentypen: Strukturen, z. B. *Adresse*
    - Datentypen entsprechen Realität: Objekte mit Verhalten

Klassen und Objekte

integrata Cegos Group
inspire qualify change

- Klasse
  - Bildet mit Eigenschaften, Methoden + Ereignissen einen Gegenstand, Prozess oder Rolle aus realer Welt nach
  - Auch.NET Framework ordnet seine Funktionalität in Klassen
  - Bauplan, nach dem sich beliebig viele Objekte herstellen lassen
- Objekt
  - Zustand
  - Instanz, Instanziieren
- Instanziieren
  - Klassen sind Referenztypen
    - Müssen im Gegensatz zu Strukturen explizit instanziiert werden
    - Schlüsselwort new: `Object o = new Object();`
  - `NullReferenceException`!
- Generische, partielle und statische Klassen

Klassen ableiten

integrata Cegos Group
inspire qualify change

- Neue Klasse
  - Ableiten von einer Basisklasse
    - Basisklasse darf **nicht** als `sealed` deklariert sein
  - Erbt alle Eigenschaften, Methoden und Ereignisse der Basisklasse
  - Erbt keine Konstruktoren
  - Implementierungsvererbung
  - Zusätzliche Mitglieder implementieren
  - Funktionalität geerbter Mitglieder durch überschreiben verändern
- Keine Mehrfachvererbung, aber beliebig viele Schnittstellen implementierbar
- Instanz
  - Schlüsselwort `this` -> Instanz Mitglieder
  - Schlüsselwort `base` -> Instanz Mitglieder der Basisklasse
- Destruktoren möglich (Garbage Collector)

Abstrakte Klassen

integrata Cegos Group
inspire qualify change

- Abstrakte Klasse
  - Liefert nur Grundriss für weitere Klassen
  - Kann selber gar nicht instanziiert werden
  - Methode ohne Implementierung -> abstrakte Methode
  - Klasse mit min. einer abstrakten Methode -> abstrakte Klasse
  - Darf nicht instanziiert werden, Versuch -> Fehlermeldung des Compilers

Eigenschaften + Felder (1/2)

- Eigenschaft
  - Enthält Werte einer Instanz
  - Implementieren am einfachsten durch öffentliches Feld
  - Stattdessen meistens privates Feld + spezielle Methoden
    - Zugriff kontrollierbar, z. B. beschränken auf Wertebereiche
    - Schreibschutz möglich
    - Nebeneffekte auslösen, z. B. konvertieren oder berechnen
    - Zugriff im Code sieht aus wie Zugriff auf öffentliches Feld
    - Zugriffsmodifizierer

- Eigenschaftendeklaration
  - Privates / geschütztes Feld
  - `get`- + `set`-Methode / Accessoren / Accessor + Mutator
  - `set`-Accessor
    - Neuer Wert = `value`
    - Weglassen = schreibgeschützte Eigenschaft

Eigenschaften + Felder (2/2)

integrata Cegos Group
inspire qualify change

- Schreibgeschützte Eigenschaft auch durch readonly-Feld
- Indexer
  - Ermöglicht Zugriff über Array-Syntax
  - 1 pro Klasse
  - Namenlose Eigenschaft
- Asymmetrische Accessoren

Methoden (1/3)

integrata Cegos Group
inspire qualify change

- Methode
  - Definiert Verhalten einer Klasse oder Struktur
  - Beliebige Anzahl von Parametern + 1 Rückgabewert (void falls nichts zurückgegeben wird)
  - Überladen
    - Mehrere Methoden gleichen Namens
    - Typ oder Anzahl der Parameter unterschiedlich
    - Compiler setzt automatisch entsprechend Parametern richtige Methode ein
    - Ersetzt Methoden mit Default-Parametern
    - C# unterstützt ab Version 4.0 auch optionale und benannte Parameter
  - generische + anonyme Methoden
  - Extension Methods

Methoden (2/3)

- Parameterübergabe
  - Werttyp
    - Kopie des Originalwerts
    - Veränderung des Parameterwerts beeinflusst nicht den Originalwert
  - Referenztyp
    - Kopie des Verweises
    - Beide Verweise referenzieren dasselbe Objekt!
    - Änderung in Methode = Änderung des Originalobjekts
  - Mit `ref` und `out` Übergabeverhalten einstellbar
  - Mit `params` Array-Parameter bzw. offene Parameterlisten definieren

Methoden (3/3)

- Vererbung
  - Zugriffsmodifizierer
  - Überschreiben
    - `virtual` in Basisklasse
    - `override` in abgeleiteter Klasse
    - `new` Neu-Implementieren bei Namensgleichheit
    - `abstract` Keine Standard-Implementierung, Überschreiben erforderlich

Konstruktoren

integrata Cegos Group
inspire qualify change

- Konstruktor
  - Versetzt Instanz einer Klasse in definierten Zustand
  - Vorteile gegenüber Methoden + Eigenschaften
  - Beim Instanziieren einer Klasse automatisch aufgerufen
  - Innerhalb der Klasse nur aus anderem Konstruktor aufrufbar
  - Initialisierer
    - `this` -> anderer Konstruktor der Klasse
    - `base` -> Konstruktor der Basisklasse
  - Parameterloser Standard Konstruktor
    - Im Quelltext normalerweise nicht sichtbar
    - Wird versteckt, sobald Klasse zusätzliche Konstruktoren implementiert
    - Kann explizit wieder implementiert werden
  - Beliebig viele Konstruktoren mit unterschiedlicher Signatur möglich

Statische Mitglieder

integrata Cegos Group
inspire qualify change

- Statische Mitglieder
  - Ohne Instanz ihrer Klasse bzw. Struktur verfügbar
  - z. B. `System.String` Klasse
    - Statische Eigenschaft `Empty`
    - Statische Methoden wie `Compare()`, `Concat()` und `Copy()`
  - Auch Felder + Ereignisse + Konstruktoren können statisch sein
  - Schlüsselwort `static`
  - Können nicht auf Instanz-Mitglieder zugreifen, kein `this` oder `base`
  - Zugriff nur über Typ, **nicht** über Instanz
  - Alle Instanzen des Typs teilen sich statische Mitglieder -> Instanz-Zähler
  - Statische Klassen

Kapitel 5

ARBEITEN MIT LOKALEN DATEIEN UND VERZEICHNISSEN

Die `File` und `FileInfo` Klassen

integrata Cegos Group
inspire qualify change

- Namespace `System.IO`
- Die `File` Klasse bietet atomare Methoden zum Lesen und Schreiben von Dateien an:
  - Zum Lesen:
    - `ReadAllText()`
    - `ReadAllLines()`
    - `ReadAllBytes()`
  - Zum Schreiben entsprechend:
    - `WriteAllXXX()`
    - `AppendAllXXX()`
- Die `File` Klasse bietet statische Elemente an wie
  - `File.Delete()`
  - `File.Exists()`
- Die `FileInfo` Klasse bietet Instanz Elemente an wie
  - `myFile.Delete()`
  - `myFile.Directory()`
  - `myFile.Exists`

Die `Directory` und `DirectoryInfo` Klassen

integrata Cegos Group
inspire qualify change

- `Directory` mit statischen Elementen
  - `Directory.Delete()`
  - `Directory.Exists()`
  - `Directory.GetFiles()`
- DirectoryInfo mit Instanz Elementen
  - `myDirectory.FullName`
  - `myDirectory.Exists`
  - `myDirectory.GetFiles()`

Die `Path` Klasse

integrata Cegos Group
inspire qualify change

- Kapselt viele I/O Funktionen
- Vorteile:
  - Weniger Code
  - Zeitersparnis
  - Konzentration auf komplexere I/O Funktionen
- Bietet u.a. folgende statischen Methoden an
  - `Path.HasExtension()`
  - `Path.GetExtension()`
  - `Path.GetTempFileName()`
  - `Path.GetTempPath()`

Kapitel 6

DATEN MIT LINQ ABFRAGEN

LINQ Einführung

integrata Cegos Group
inspire qualify change

- LINQ = **L**anguage **In**tegrated **Q**uery
- Namespace `System.Linq`
- LINQ ist
  - Standardisiert
    → Die Syntax bleibt gleich, egal welche Datenquelle man abfragt
  - Deklarativ
    → Programmierkonzept, das beschreibt was man tut möchte, ohne beschreiben zu müssen wie es getan wird
- Ähnelt klassischer SQL Syntax

LINQ verwenden

- LINQ kann entweder als Ausdruck verwendet werden:

```
var ergebnis = from element in sammlung
               orderby element.Eigenschaft1 ascending
               select element;
```

- Oder als Erweiterungsmethode:

```
var erstesElement = sammlung.FirstOrDefault();
var anzahl = sammlung.Count();
```

Nachbesprechung

LINQ

Kapitel 7

WIEDERVERWENDBARE TYPEN UND ASSEMBLIES

Was ist Reflection

integrata Cegos Group
inspire qualify change

- Namespace `System.Reflection` bietet folgende Typen
  - `Assembly`
  - `TypeInfo`
  - `ParameterInfo`
  - `ConstructorInfo`
  - `FieldInfo`
  - `MemberInfo`
  - `PropertyInfo`
  - `MethodInfo`

Auslesen von Assemblies

integrata Cegos Group
inspire qualify change

- `Assembly.GetExecutingAssembly()`
- `Assembly.Load("NameDerAssembly")`
- `Assembly.LoadFile(@"C:\Pfad\zur\Assembly.dll")`
- `Assembly.LoadFrom("NameOderPfadDerAssembly")`
- `Assembly.ReflectionOnlyLoad("NameDerAssembly")`
- `Assembly.ReflectionOnlyLoadFrom(@"C:\Pfad\zur\Assembly.exe")`

Auslesen von Typen


```
var asm = Assembly.GetExecutingAssembly()
asm.GetTypes()
asm.GetType("…")
```

- `typeof(Program).GetConstructors();`
- `typeof(Program).GetFields();`
- `typeof(Program).GetProperties();`
- `typeof(Program).GetMethods();`

Instanziieren eines Typs

- Namespace `System`
- `Activator.CreateInstance(Type type)`
- `Activator.CreateInstance(Type type, params object[] args)`
- `CreateInstance()` gibt ein `ObjectHandle` zurück, dass erst noch umgewandelt werden muss:
  `(ISomeType)Activator.CreateInstance(myType)`
  ➔ Entspricht einer Instanziierung mit `new`

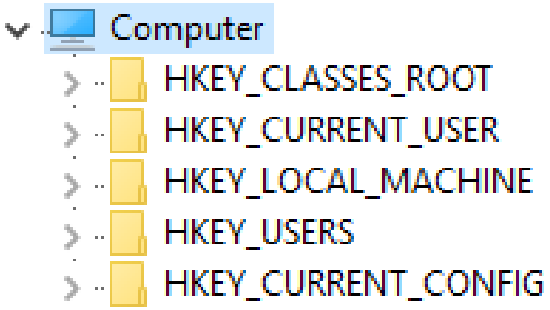
Kapitel 8

ARBEITEN MIT DER WINDOWS REGISTRIERUNG

Windows Registrierung (1/2)

integrata Cegos Group | inspire qualify change

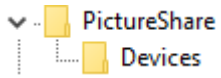
- Windows Registrierung besteht aus mehreren Hauptbereichen



- HKEY steht für **H**andle to **Key**
- Entsprechend werden Hauptzweige gewöhnlich abgekürzt, z.B:
  - HKLM ➔ **H**andle to **Key** **L**ocal **M**achine
  - HKCU ➔ **H**andle to **Key** **C**urrent **U**ser
- In vielen Fällen wird in `CURRENT_USER` und/oder `LOCAL_MACHINE` gearbeitet

Windows Registrierung (2/2)

- Die einzelnen Elemente werden wie folgt bezeichnet:



→ Schlüssel

Name	Typ	Daten
(Standard)	REG_SZ	(Wert nicht festgelegt)
AutoloadMenu ❶	REG_SZ ❷	1 ❸
LocalPath	REG_SZ	C:\Users\ThomasOhms\Pictures\PictureShare

❶ Wert
❷ Wertetyp
❸ Daten

- Die Registrierung unterstützt folgende Wertetypen:
  - Zeichenfolge
  - Binärwert
  - DWORD-Wert (32 bit)
  - QWORD-Wert (64 bit)

Die `Registry` Klasse (1/2)

integrata Cegos Group
inspire qualify change

- Namespace `Microsoft.Windows32`
- `Registry` Klasse bietet 7 statische Felder als abgekürzte Variante zu den einzelnen Hauptschlüsseln:
  - `ClassesRoot`
  - `CurrentConfig`
  - `CurrentUser`
  - `DynData`
  - `LocalMachine`
  - `PerformanceData`
  - `Users`
- Der Benutzer unter dem eine Anwendung ausgeführt wird und die Berechtigungen der einzelnen Schlüssel, entscheiden darüber welche Bereiche les-, bzw. beschreibbar sind

Die `Registry` Klasse (2/2)

integrata Cegos Group | inspire qualify change

- Die Registry Klasse bietet Methoden zum lesen, als auch schreiben eines Eintrags an:
  - `public static object GetValue(string keyName, string valueName, object defaultValue)`
  - `public static void SetValue(string keyName, string valueName, object value)`
  - `public static void SetValue(string keyName, string valueName, object value, RegistryValueKind valueKind)`
- `RegistryValueKind` ist eine Auflistung, die den Typ des zu speichernden Wertes angibt

Die Klasse `RegistryKey`

integrata Cegos Group | inspire qualify change

- Mit `RegistryKey` hat man die Möglichkeit Unterschlüssel zu öffnen und zu erstellen
  - `OpenSubKey`
  - `CreateSubKey`
- **`RegistryKey` implementiert `IDisposable`!**

- Ein simples Beispiel:

```
var hauptschlüssel = Registry.CurrentUser;
var software = hauptschlüssel.OpenSubKey("Software");
var meinSchlüssel = software.CreateSubKey(@"MeinProgramm/Unterschlüssel", true);

meinSchlüssel.SetValue("Wertname", "Daten", RegistryValueKind.String);
meinSchlüssel.GetValue("Wertname");

meinSchlüssel.Dispose();
```

Kapitel 9

WINDOWS FORMS

Zweck einer Windows-Anwendung

integrata Cegos Group
inspire qualify change

- Für komplexe Oberflächen im Look & Feel von Windows
- Windows auf 90 % der Client-Rechner → Bedienungssystematik bekannt
- Andere BS / UI gleichwertig, aber Spezialanwendungen oft nur für Windows + Portierung für Hersteller unwirtschaftlich → BS-Migration unwahrscheinlich
- Viele Steuerelemente im Betriebssystem eingebaut → Baukastensystem
- Microsoft nutzt Namensraum `System.Windows.Forms`, andere .NET-Implementierungen bieten evtl. andere Oberflächen-APIs → hier am ehesten Inkompatibilitäten zw. unterschiedlichen .NET-Implementierungen
- Windows-Anwendung nicht Voraussetzung für Zugriff auf Windows-API

Windows-Anw. in VS erstellen

integrata Cegos Group | inspire qualify change

- Vorlage Windows-Anwendung in Visual Studio
- `Form1.cs`, Name änderbar
- `Form1.Designer.cs`, wenn alle Dateien sichtbar
- Mit F7 / Umsch+F7 zwischen Designer + Editor umschalten
- `Form1.cs`:
  - Namensraum = Projektname != Klassenname, evtl. `Run()` anpassen
  - Klasse `Form1` mit Methoden + Feld, von Form abgeleitet
  - Eigene Felder + Definitionen auf Klassenebene oberhalb Konstruktor
  - Partielle Klasse

Funktionsweise einer Windows-Anw.

integrata Cegos Group | inspire qualify change

- Start → `Main()` in `Program.cs`
- Ende `Main()` → Ende Programm
- Bei mehreren Formularen Startobjekt festlegen
- `Main(): args`-Parameter und `int` statt `void` möglich
- `Application.Run()` → Nachrichtenschleife
- `InitializeComponent()` konfiguriert Formular + Steuerelemente und fügt sie `Controls`-Auflistung hinzu
- `InitializeComponent()` automatisch generiert → zusätzliche Initialisierungsfunktionalität nur in eigener Methode
- `Dispose()` beim Freigeben des Formulars automatisch aufgerufen, um zusätzlichen Freigabe-Code erweiterbar

Oberflächengestaltung

integrata Cegos Group
inspire qualify change

- Oberfläche einer Windows-Anwendung
  - Besteht meistens aus vorhandene Steuerelementen
  - Selten durch grafische Operationen erzeugt

- Diverse Registerkarten in Toolbox
  - Erweiterbar
  - Nur in Designer-Ansicht verfügbar

Einfache Steuerelemente einsetzen

integrata Cegos Group
inspire qualify change

- Label, Schaltfläche, Textfeld, Kontrollkästchen, Listen- und Kombinationslistenfeld kennt jeder
- Erben die meisten Methoden von `Component`– oder `Control`-Klasse
- Textfeld
  - `AppendText()`, `Clear()`, `Cut()` und `Undo()` zur Textbearbeitung
  - `Text`-Eigenschaft + String-Array `Lines`
- Kombinationslistenfeld
  - `FindString()` und `GetItemText()` zur Arbeit mit seiner Liste
  - `Items`-Auflistung
  - `DropDownStyle` und `Sorted` zur Konfiguration der Liste
  - `AutoCompleteMode`, `AutoCompleteSource`
- Container-Steuerelemente

Steuerelem. zur Laufzeit ausrichten lassen

integrata Cegos Group
inspire qualify change

- Zur Laufzeit:
  - Größenänderungen des Fensters verhindern
    → einfach für Programmierer, nervig für Benutzer
  - Fenster-Layout den Größenänderungen dynamisch anpassen
- Dynamisches Anpassen des Layouts:
  - Visual Basic: Code schreiben / Zusatzsteuerelemente kaufen
  - Java: Layout-Manager
  - .NET: `Anchor`- + `Dock`-Eigenschaften → schnell + einfach + angenehm
  - `FlowLayoutPanel`, `TableLayoutPanel` = Layout-Manager
- `Dock` bindet Steuerelement an innere Kante des Container-Steuerelements
- `Anchor` behält Abstand zu einer oder mehreren Kanten bei
- `Dock` + `Anchor` kombinierbar, Container-Steuerelemente schachtelbar
  → beliebige Layout-Regeln möglich

Menüs einsetzen (1/4)

integrata Cegos Group
inspire qualify change

- Menüs bieten
  - Systematischen Zugriff auf gesamte Funktionalität
  - Überblick verfügbarer Funktionalität
- Hauptmenü unterhalb der Titelleiste
  - Menü: Eintrag in Menüleiste + Menübefehle darunter
  - Gruppen von Menübefehlen oft durch Strich voneinander getrennt
- Kontextmenü:
  - Untermenge von Befehlen des Hauptmenüs
  - Untermenüs in Befehlen eher unbeliebt

Menüs einsetzen (2/4)

integrata Cegos Group
inspire qualify change

- `MenuStrip`- + `ContextMenuStrip`-Steuerelemente → Komponentenfach
- Formular-Designer blendet automatisch Menü-Designer in Menüleiste des Formulars ein
- Befehlstext eingeben + durch `&`-Zeichen evtl. Zugriffstaste festlegen
- Weitere Menüs + Menübefehle durch Ausfüllen der *Hier eingeben*-Felder erzeugen
- Bestehende Menüs + Menübefehle durch Drag & Drop verschieben
- Weitere wichtige Eigenschaften
  - `CheckOnClick`-Eigenschaft
  - `RadioCheck`-Eigenschaft entfällt
  - `Image`-Eigenschaft
  - Befehl Standardelemente einfügen!

Menüs einsetzen (3/4)

integrata Cegos Group
inspire qualify change

- Eigenschaften einstellen
  - Menüstruktur schaffen
  - Eigenschaften der `ToolStripMenuItems` im Eigenschaften-Fenster einstellen
    - Name: hierarchische Konvention für Namensaufbau (z.B. mnuDateiBeenden)
    - Tastaturkürzel: `ShortCut`, `ShowShortCut`
    - Befehlstext: Text, &-Zeichen / nur Bindestrich

Menüs einsetzen (4/4)

- Ereignisse behandeln
  - Hauptsächlich `Click`-Ereignis; Doppelklicken -> neuer Ereignishandler
  - `DropDownOpening`-Ereignis: Menü wird gleich gezeigt, z. B. für dynamisch Zusammenstellen / Konfigurieren

Standarddialoge einsetzen

integrata Cegos Group
inspire qualify change

- Standardisierte Fenster für bestimmte Aufgaben
- Registerkarte Dialogfelder der Toolbox
- Methoden
  - `Reset()` um Eigenschaften auf Standardwerte zurückzusetzen
  - `ShowDialog()` zum Anzeigen des Fensters
  - `OpenFileDialog`- und `SaveFileDialog`-Standarddialoge
    - `OpenFile()`-Methode für Stream-Zugriff auf gewählte Datei
- Statt Standarddialog-Steuerelement oft nur für Benutzung instanziieren
  - `Dispose()` aufrufen oder `using`-Anweisung benutzen
- Lösungsmuster
  - Durch Eigenschaften konfigurieren
  - `ShowDialog()`-Methode aufrufen
  - Bei `DialogResult.OK` die Auswahl übernehmen

Spezifische Formular-Eigenschaften

integrata Cegos Group | inspire qualify change

- `AcceptButton` / `CancelButton`:
  - Schaltfläche auswählen
  - Enter- / Esc-Taste löst Schaltfläche aus
  - `DialogResult` != `DialogResult.None` schließt modales Fenster
- `ShowInTaskbar`, `StartPosition`, `TopMost`, `WindowState`
- `TransparencyKey`, `BackgroundImage`, `FormBorderStyle.None`

Spezifische Formular-Ereignisse

integrata Cegos Group
inspire qualify change

- `Activated` / `Deactivate` statt `Enter` / `Leave`
- `Load`, `FormClosing` (Cancel), `FormClosed`
  → Ereignisreihenfolge s. Online-Hilfe
- `Paint`: Parameter Graphics-Objekt

Spezifische Formular-Methoden

integrata Cegos Group | inspire qualify change

- Hauptformular automatisch erzeugt + gezeigt
- Zusätzl. Formulare selbst instanziieren + mit `Show()` / `ShowDialog()` zeigen
- `Show()` → nicht-modales Fenster: beliebiger Fensterwechsel
- `ShowDialog()` → modales Fenster: vor Wechsel schließen
- Keine System-modalen Fenster
- Nicht-modales Fenster:
  - Mit `Close()` oder Systemschaltfläche schließen
  - Mit `Hide()` verstecken
- Modales Fenster:
  - `Close()` oder Systemschaltfläche verstecken nur!
  - Öffentliche Eigenschaften noch verfügbar
  - `ShowDialog()` blockiert aufrufende Methode
  - Mit `Dispose()` freigeben

Kapitel 10

SERIALISIERUNG

Unterstützte Formate

integrata Cegos Group | inspire qualify change

- .NET unterstützt 3 unterschiedliche Formate der Serialisierung
  - Binär

```
10101010101011111010110101010110101111111010101101100001
```

  - XML

```
<SOAP-ENV:Body>
        <a1:MeinSerialisierbarerTyp id="ref-1"
                xmlns:a1="NAMESPACE+ASSEMBLYNAME+VERSION">
                <SomeInt>1</SomeInt>
                <SomeString id="ref-3">Irgendein Text</SomeString>
        </a1:MeinSerialisierbarerTyp>
</SOAP-ENV:Body>
```

  - JSON

```
{
   "Eigenschaft1":"Wert1",
   "Eigenschaft2":"Wert2"
}
```

Beispiel eines eigenen, serialisierbaren Typs

integrata Cegos Group | inspire qualify change

```
[Serializable]
public class MeinSerialisierbarerTyp : ISerializable
{
    public MeinSerialisierbarerTyp()
    { }

    protected MeinSerialisierbarerTyp(SerializationInfo info,
                                      StreamingContext context)
    {}

    public virtual void GetObjectData(SerializationInfo info,
                                      StreamingContext context)
    {}
}
```

Die "Formatierer"

integrata Cegos Group
inspire qualify change

- `BinaryFormatter`
- `SoapFormatter`
- `DataContractJSONSerializer`
- `IFormatter`
- `ISerializable.Serialize()`

- Ablauf der `Serialize()` Methode
  1. Formatter prüft, ob Surrogate Selector existiert
     1. Wenn ja, Prüfung, ob dieser Objekte des übergebenen Typs behandelt und Ausführung von `ISerializable.GetObjectData()`
     2. Wenn nein oder der Typ nicht behandelt wird → Suche nach `Serializable` Attribut
        1. Existiert dieses Attribut nicht ➔ `SerializationException`
        2. Existiert das Attribut, erneut Prüfung auf `ISerializable` → Ausführung von `GetObjectData()`
        3. Existiert das Attribut, `ISerializable` aber nicht, wird Standard-Serialisierung für **alle** Eigenschaften genutzt, die **nicht** als `NonSerialized` gekennzeichnet wurden

Eigene "Formatierer"

integrata Cegos Group
inspire qualify change

```
class CsvFormatter : IFormatter
{
    public ISurrogateSelector SurrogateSelector { get; set; }
    public SerializationBinder Binder { get; set; }
    public StreamingContext Context { get; set; }

    public object Deserialize(Stream serializationStream)
    {
    }

    public void Serialize(Stream serializationStream, object daten)
    {
    }
}
```

Kapitel 11

ADO.NET

Aufbau von ADO.NET

integrata Cegos Group
inspire qualify change

- Verwaltete Provider
  - Auf bestimmtes DBMS oder Datenzugriffstechnologie spezialisiert
  - Stellen Verbindung zur DB her
  - Lesen und manipulieren Daten über SQL-Anweisungen
- `DataSet`- und `DataTable`-Klassen
  - Unabhängig von DBMS oder Datenzugriffstechnologie
  - Verwalten Daten auf dem Client
  - Speichern Änderungen zunächst lokal
  - Können Daten im XML-Format speichern
- Web- oder Windows-Steuerelemente
  - Binden beliebige Eigenschaften an Datenmenge
  - Datenmenge kann auch Array o.ä. sein

Verwaltete Provider (1/6)

integrata Cegos Group
inspire qualify change

- .NET: Provider für SQL Server (Compact), Oracle, OLE DB, ODBC
- Drittanbieter: z. B. Provider für Firebird oder MySQL
- Normalerweise verwalteter Code
  → Ausnahme: Legacy
- Keine unnötigen Schichten, möglichst nur .NET-Provider benutzen

Verwaltete Provider (2/6)

- Verwaltete Provider bestehen aus
  - `Connection`-Objekt für DB-Verbindung
  - Datenlese-Objekt
  - Befehlsobjekte, repräsentieren SQL-Anweisungen
  - Datenadapter-Objekt zum Verbinden mit DataSet-Objekt

- Jeder verwaltete Provider
  - Benutzt anderes Präfix: Sql, Oledb, Odbc, usw.
  - Implementiert gleiche Schnittstellen: IDbConnection, etc.
  - Trotz anderer Klassennamen gleiche Eigenschaften + Methoden

Verwaltete Provider (3/6)

integrata Cegos Group | inspire qualify change

- Die `SqlConnection`-Klasse
  - `ConnectionString`
  - `Open()` / `Close()`
  - `ConnectionTimeout`, `PacketSize`
  - `BeginTransaction()` → `SqlTransaction`, `Commit()`, `Rollback()`
  - `Close()` / `Dispose()`

Verwaltete Provider (4/6)

integrata Cegos Group | inspire qualify change

- Namespace `System.Data.SqlClient`
- Die `SqlCommand`-Klasse
  - `CommandText`
    - Transact-SQL
    - Tabellenname
    - Gespeicherte Prozedur
  - `CommandType`
  - `CommandTimeout`
  - `Parameters` → evtl. auch benannte
  - `Connection`, `Transaction`
  - `ExecuteNonQuery()` → Anzahl betroffener DS
  - `ExecuteReader()` → SqlDataReader
  - `ExecuteScalar()` → Einzelwert
  - `Dispose()`

Verwaltete Provider (5/6)

integrata Cegos Group
inspire qualify change

- Die `SqlDataReader`-Klasse
  - Schnell, aber nur lesen + nur vorwärts
  - `GetInt32()`, `GetString()` → `IsDBNull()` nicht vergessen!
  - `Read()` → nächster DS
  - Erzeugung über `SqlCommand.ExecuteReader()`
  - `Close()`, macht `SqlConnection` wieder verfügbar

Verwaltete Provider (6/6)

integrata Cegos Group | inspire qualify change

- Die `SqlDataAdapter`-Klasse
  - Navigierbare Datenmenge, benutzt `SqlDataReader`-Klasse
  - `SelectCommand` = SELECT + `SqlConnection`
  - `Fill()` füllt `DataTable`
  - `Update()` speichert `DataTable`
    - Sendet für jeden DS 1 SQL-Anweisung an DB
      - UpdateBatchSize für SQL Server
    - Braucht dazu `DeleteComand`, `InsertCommand`, `UpdateCommand`
    - `SqlCommandBuilder` machts einfach
      - `SetAllValues`-Eigenschaft
      - `ConflictOption`-Eigenschaft
- `TableAdapter`-Klasse

Datenbankunabhängige Klassen (1/4)

integrata Cegos Group | inspire qualify change

- Namespace `System.Data`
- Die `DataColumn`-Klasse
  - Beschreibt Tabellenspalte:
    - `AllowDBNull`, `AutoIncrement`, `ColumnName`, `DataType`, `ReadOnly`, `Unique`, etc.
  - In `Columns`-Auflistung des `DataTable`-Objekts verwaltet
  - Beim Füllen des `DataTable`-Objekts automatisch erzeugt
  - Programmgesteuertes Erzeugen möglich

Datenbankunabhängige Klassen (2/4)

integrata Cegos Group | inspire qualify change

- Die `DataRow`-Klasse
  - Enthält Felder eines Datensatzes
  - Zugriff auf Feldwerte
    - `Item`, `ItemArray`
    - Datentyp = `System.Object`
    - `DataRowVersion.Current`, `DataRowVersion.Original`
    - `DataRowState.Modified`, `DataRowState.Unchanged`
  - In `Rows`-Auflistung des `DataTable`-Objekts verwaltet
  - Datensatz anlegen: `NewRow()` + `Rows.Add()`
  - Datensatz ändern: Indexer, evtl. `BeginEdit()` + `EndEdit()`
  - Datensatz löschen: `Delete()`, **nicht** `Remove()`
  - Zurückschreiben in DB i. A. mit `Update()` eines Datenadapters

Datenbankunabhängige Klassen (3/4)

integrata Cegos Group
inspire qualify change

- Die `DataTable`-Klasse
  - Repräsentiert Datemenge
  - Füllen mit `Fill()` eines Datenadapters
  - Programmgesteuertes Füllen möglich
  - `Columns`- und `Rows`-Auflistung, `Constraints`
  - Zurückschreiben in DB i. A. mit `Update()` eines Datenadapters
    - Problematisch bei mehreren Tabellen!
    - `AcceptChangesDuringUpdate`-Eigenschaft

Datenbankunabhängige Klassen (4/4)

integrata Cegos Group
inspire qualify change

- Die `DataSet`-Klasse
  - Verwaltet mehrere Datemengen
  - `Tables`- und `Relations`-Auflistung
    → Kann ganze relationale DB aufnehmen
  - Speichern im XML-Format mit `WriteXml()`
    - Daten + Schema
    - Daten + Änderungen
    - Nur die Änderungen: `GetChanges()` + `WriteXml()`

ADO.NET-Unterstützung in Visual Studio (1/3)

integrata Cegos Group | inspire qualify change

- Rapid Application Development
- Oberfläche generieren lassen
- Features in Visual Studio
  - Datenquellen-Fenster
  - Assistenten zum Konfigurieren von Datenquellen
  - Z. B. Tabelle auf Formular ziehen
    ➔ Benötigte Steuerelemente entstehen automatisch!
  - `DataGridView` ersetzt `DataGrid`
  - `BindingNavigator`
  - Generierte Komponenten auf Registerkarte Datenquellen Komponenten verfügbar
  - `DataSet`-Designer
  - Datenbankdateien werden kopiert

ADO.NET-Unterstützung in Visual Studio (2/3)

integrata Cegos Group | inspire qualify change

- Typisiertes DataSet
- Vorher / nachher:

```
// DataSet:
string nachname = dsNamen.Tables["Namen"].Rows[idx]["Nachname"].ToString();


// Typisiertes DataSet:
string nachname = dsNamen.Namen[idx].Nachname;
```

- Vorteile
  - Code ist verständlicher
  - Typsichere Eigenschaften ersparen
    - ständige Typumwandlung
    - Laufzeitfehler durch Tippfehler im Eigenschaftennamen
  - Visual Studio kann IntelliSense und Code-Vervollständigung bieten
- Aufbau
  - `BindingSource`
  - `TableAdapter`
  - …

ADO.NET-Unterstützung in Visual Studio (3/3)

integrata Cegos Group | inspire qualify change

- Datenbindung der Steuerelemente
  - Steuerelemente zeigen autom. aktuelle Daten
  - Änderungen werden autom. **Lokal (!)** gespeichert
  - Voraussetzung für Datenbindung zur Designzeit
    - Konfigurierte `DataSet`-Komponente im Projekt
  - Einfache Datenbindung
    - Ein Feld im aktuellen Datensatz
    - Nicht nur `Text`, `Checked`, … bindbar
    - Z. B. Vorder- und Hintergrundfarbe aus DB steuern
  - Komplexe Datenbindung
    - Mehrere Felder, mehrere Datensätze
    - Z. B. `GridView`
    - Listen- + Kombinationslistenfeld: einfache + komplexe Bindung

Entity Framework

integrata Cegos Group
inspire qualify change

- Objektrelationaler Mapper (ORM)
  - Mapping eines Datenbankmodells in ein Objektmodell
  - Lesende und schreibende Zugriffe
  - Kein SQL mehr im Programmcode
    → Compilerfehler, statt Laufzeitfehler wegen strenger Typisierung
  - Vollständige Intellisense auf **alle** Datenbankobjekte

- Zugriff mittels LINQ to Entities
  - Eingeführt mit dem .NET Framework 3.5

Kapitel 12

.NET UND XML

Die XML-Infrastruktur (1/2)

integrata Cegos Group
inspire qualify change

- XML-Infrastruktur unter .NET
  - Namensraum `System.Xml`
    - Vollständige DOM-Implementierung
    - Erweiterungen
    - Kann gegen DTDs oder XML-Schema validieren
  - Namensräume `System.Xml.Xsl` + `System.Xml.XPath`
    - Klassen zum Transformieren von XML-Dokumenten
  - `DataSet` → `XmlDataDocument` → XML (auch zurück)

Die XML-Infrastruktur (2/2)

integrata Cegos Group | inspire qualify change

- XML-Infrastruktur unter .NET
  - `XmlReader`-Klasse
    - .NET-Spezialität
    - Lesen mit `Read()`, `Skip()`, … steuern
    - Auch von DOM-Implementierung genutzt
    - Keine Navigations- oder Manipulationsmöglichkeit
    - Kein Einsatz von XPath-Ausdrücken
    - Ersetzt `XmlTextReader`- + `XmlValidatingReader`-Klasse
      - Standardkonformer
      - Erzeugen mit Factory-Methode
      - Konfigurieren mit `XmlReaderSettings`-Instanz
  - Schreiben mit `XmlWriter`-Klasse

- XML-Dokumente mit DOM erzeugen
  - Klassen: `XmlDocument`, `XmlElement`, `XmlAttribute`, …
  - Dokumentbaum aus Objekten aufbauen
  - Elemente z. B. nachträglich um Attribute erweiterbar
  - Transformation durchführbar
  - Element / Attribut anlegen
    - 1: Instanz durch Factory-Methoden der `XmlDocument`-Instanz erzeugen
    - 2: Instanz ihrem übergeordneten Element / dessen Attributliste hinzufügen
  - Element + Textinhalt durch `CreateTextNode()` in einem Schritt erzeugen
  - Vereinfachung existiert für Attribute leider nicht
  - XML als String in einem Schritt setzen
    - `XmlDocument: LoadXml()`
    - `XmlNode` / `XmlElement: InnerXml`

Die XML-Infrastruktur unter .NET nutzen (3/8)

integrata Cegos Group | inspire qualify change

- XML-Dokumente mit `XmlWriter`-Klasse erzeugen
  - Eher text- als objektorientiert
  - Stellt nur rudimentär die Wohlgeformtheit des generierten XML sicher
  - Nachträglich Elemente / Attribute einfügen nicht möglich
  - Transformationen mangels Dokumentbaum nicht möglich
  - Vorteil: Ausgabe besser konfigurierbar als bei reinen DOM-Klassen
  - Wenn Element untergeordnete Elemente oder Attribute besitzt
    - `WriteStartElement()`, `WriteEndElement()`, evtl. `WriteString()`
  - Sonst `WriteElementString()`

Copyright und Impressum

Integrata AG
Zettachring 4
70567 Stuttgart